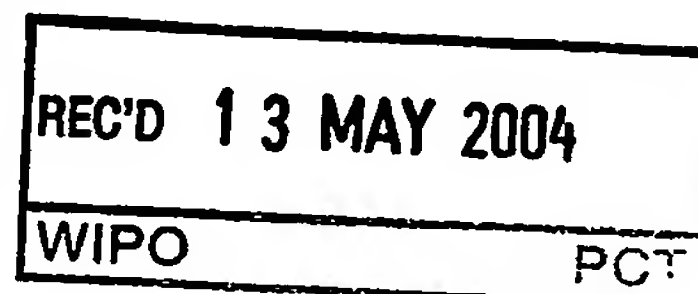


29. 04. 2004



**Prioritätsbescheinigung über die Einreichung
einer Patentanmeldung**

Aktenzeichen: 103 14 832.9

Anmeldetag: 01. April 2003

Anmelder/Inhaber: Siemens Aktiengesellschaft, 80333 München/DE

Bezeichnung: Verfahren und Anordnung zur kundenseitigen
Anpassung von Software

IPC: G 06 F 9/45

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 23. April 2004
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Agurks

**PRIORITY
DOCUMENT**
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH RULE 17.1(a) OR (b)

Beschreibung

Verfahren und Anordnung zur kundenseitigen Anpassung von Software

5

Die Erfindung betrifft ein Verfahren und eine Anordnung zur Transformation von Software, bei dem/der eine Software, in eine Darstellung in einer Meta-Auszeichnungssprache, beispielsweise XML, übergeführt, dort, beispielsweise mit XSLT, transformiert und dann diese in der Meta-Auszeichnungssprache formulierte transformierte Darstellung in eine modifizierte Software, beispielsweise derselben Ausgangssprache, zurückverwandelt wird.

15 Aus dem Internet ist unter <http://beautyj.berlios.de/> ein Java Source Code Transformation Tool BeautyJ bekannt, bei dem ein Java Quellcode in eine XML-Darstellung umgewandelt wird, mittels Sourclet API, beispielsweise durch Einfügen von Leerzeichen oder geänderten Kommentaren an bestimmten Stellen, „verschönert“ und anschließend der modifizierte Quellcode in Java Quellcode zurück konvertiert werden kann. Eine Transformation mittels XSLT wird hier, für diesen Zweck, nur vorgeschlagen, aber nicht umgesetzt.

20

Die der Erfindung zugrunde liegende Aufgabe liegt nun darin, ein Verfahren und eine Anordnung zur Modifikation von Quellcode anzugeben, bei dem/der eine weitergehende noch flexiblere und effizientere Modifikation der Quellcodes erreicht wird.

30

Diese Aufgabe wird hinsichtlich des Verfahrens durch die Merkmale des Patentanspruchs 1 und hinsichtlich der Anordnung durch die Merkmale des Anspruchs 6 erfindungsgemäß gelöst. Die weiteren Ansprüche betreffen bevorzugte Ausgestaltungen der Erfindung.

35

Die Erfindung besteht im Wesentlichen darin, dass ausgewählte Bestandteile einer Software, Klassen und/oder einzelne Fragmente, als Variationspunkte VP dienen können, indem diese in einen in einer Meta-Auszeichnungssprache formulierten ersten Code (CodeML) umgewandelt werden, die Software nun in Mischform, d.h. kompilierter Code und CodeML, ausgeliefert wird, und der erste Code CodeML kundenseitig durch eine oder mehrere Transformationen T, bspw. XSLT, nur in Abhängigkeit von Transformationsregeln TR, die Modifizierungsregeln für die Variationspunkte VP enthalten, in einen in der Meta-Auszeichnungssprache formulierten zweiten Code CodeML* umgewandelt werden kann und dieser zweite Code wiederum gleich als Variationspunkt oder in kompilierter Form die Abarbeitungsreihenfolge und/oder das Verhalten der Software verändert, woraus folgt, dass sich die erste und die zweite Software an den Variationspunkten unterscheiden.

Die Erfindung wird im Folgenden anhand des in Zeichnung 1 dargestellten Beispiels näher erläutert. Dabei zeigt

Zeichnung 1 ein Gesamtblockdiagramm zur Erläuterung der Erfindung.

In Zeichnung 1 ist ein Gesamtblockdiagramm zur Erläuterung der Erfindung dargestellt, bei dem zunächst eine Software SW bestehend aus Quelltext SC1, SC2 und SC in eine auslieferungsfähige Software SW* umgewandelt wird, wobei einige Teile der Software wie bspw. SC1 nun als Binärcode/Byte Code B1 zur Verfügung stehen, und andere Teile wie bspw. SC2 durch einen Konverter CONV in einen in einer Meta-Auszeichnungssprache formulierten ersten Code CodeML umgewandelt werden, so daß sie in der lauffähigen Software SW* fortan Variationspunkte VP bspw. VP1 bilden. Diese Software SW* kann vor/oder zur Laufzeit auf eine Weise modifiziert werden, dass der in der Meta-Auszeichnungssprache dargestellte Code VP, bspw. VP2, mit einer Transformation T und Transformationsregeln TR in einen zweiten in der Meta-

Auszeichnungssprache formulierten Code **CodeML*** umgewandelt wird, welcher nun entweder als geänderter Variationspunkt bspw. **VP2*** in **SW*** vorhanden ist oder durch einen Konverter **RCONV** nach der Transformation **T** in einen Quellcode **SC*** und danach mittels **COMP** in einen ByteCode/Binärcode **VP2B*** überführt wird. In beiden Fällen unterscheiden sich **SW** und **SW*** an den Stellen der Variationspunkte und können auf diese Weise an spezifische Anforderungen (bspw. Toolkit-Austausch, Updates, usw.) angepasst werden.

Die Codes **CodeML** und **CodeML*** bzw. **VP** und **VP*** sind beispielsweise in der Meta-Auszeichnungssprache XML formuliert, wobei „XML“ für Extended Markup Language steht.

Von besonderem Vorteil ist hierbei, dass dies nicht vom Programmentwickler durchgeführt werden muss, sondern vom entsprechend ausgestatteten und informierten Kunden selbst erledigt werden kann. Hierzu braucht ein Operator oder Administrator auf der Kundenseite nur eine entsprechende Transformation **T** mit den benötigten Ersetzungs-, Modifikations- und Entfernungsregeln **TR** anzuwenden, um die Software auf seine speziellen Bedürfnisse anzupassen bzw. ein Update oder Patching durchzuführen. Beim Update oder Patching von kundenspezifisch angepasster treten bislang häufig Probleme wegen Inkonsistenzen auf, die durch diese Erfindung und die Möglichkeit der Pipelineanwendung bzw. geordnete Hintereinanderausführung von Anfang an vermieden werden können.

Die im **Anhang** befindlichen Programmauflistungen **Listing 1** bis **Listing 5** zeigen dies an einem konkreten Beispiel: Typischerweise kann eine Software-Auslieferung an zwei unterschiedliche Kunden unterschiedliche Toolkits verwenden, die sich hinsichtlich Performance, Preis usw. unterscheiden.

So soll hier ein Code der ursprünglich eine Registrierungsklasse

import electric.registry.Registry
 aus einem Glue-Toolkit verwendet, beim zweiten Kunden nun
 zwei neue "Registrierungsklassen"
 import org.apache.axis.client.Call und
 5 import org.apache.axis.client.Service aus einem Axis-Toolkit
 verwenden.

In XSL kann dies z.B. mittels

```

10 <xsl:template match="import">
    <xsl:if test="dot/name='Registry'">
        <import>
            <dot>
                <dot><dot><name>org</name><name>apache</name></dot><name>axis</name></dot>
                <name>client</name></dot><name>Call</name>
            </dot>
        </import>
        <import>
            <dot>
20         <dot><dot><name>org</name><name>apache</name></dot><name>axis</name></dot>
                <name>client</name></dot><name>Service</name>
            </dot>
        </import>
    </xsl:if>
25 <xsl:if test="dot/name!='Registry'">
    <xsl:copy-of select="."/>
    </xsl:if>
    ...
    </xsl:template>
30
```

geschehen. Schablonen bzw. Templates werden in XSL auf das in
 match definierte Muster angewendet. Das import-Template im
 Listing-Beispiel also auf alle ursprünglichen import-
 Anweisungen. Im konkreten Beispiel ignoriert es einfach alle
 ursprünglichen Glue-Registry-Imports, und fügt stattdessen
 die zwei Axis-spezifischen imports ein.

Durch das erfindungsgemäße Verfahren ergeben sich noch eine
 Reihe von zusätzlichen Vorteilen, wie beispielsweise:

40

1. Es ist nur ein System für Problemstellungen wie Patching,
 Customizing, Updating, etc. erforderlich und nicht eine Reihe
 verschiedener teilweise proprietärer Werkzeuge.

45

2. Das Verfahren basiert auf Standards wie XML und XSLT und
 ist hinsichtlich der Konvertierbarkeit in andere

Programmiersprachen geringeren Beschränkungen unterworfen als andere Verfahren zur Modifikation von Quellcode.

3. Selbst für spezielle und komplizierte Quellcode-
5 Modifikationen sind keine proprietären Speziallösungen erforderlich, sondern es können hierfür existierende Standards wie **XSLT**, **XPath** und **XQuery** genutzt werden.

10 4. Diese Art der Modifikation erlaubt die Erstellung von Hierarchien u.a. durch die Möglichkeit zur geordneten, automatisierten Hintereinanderausführung (Pipelines) mehrerer Transformationen, bspw. von Patches.

15 5. Die Transformationen können für eine allgemeine Wiederverwendung in XSLT-Dateien gespeichert werden, so daß Bibliotheken z.B. für bestimmte Abläufe entstehen können.

20 6. Es kann eine XML-Repräsentation des Quellcodes in einer XML-Datenbasis gespeichert und bei Bedarf mit Hilfe einer XSLT in einfacher Weise an die jeweiligen Kundenbedürfnisse angepasst werden (Customization).

7. Durch die Verwendung der Standards **XMLSchema** oder **DTD** oder entsprechende XSLTs kann der Code vorab (ohne Kompilierung), auf bestimmte Korrektheitsaspekte hin, überprüft (validiert) werden.

30 8. Übliche XML-Tools können zur einfachen Bearbeitung bzw. Visualisierung und Bestimmung von Beziehungen im Code verwendet werden.

35 9. Dauerhafte XML-basierte Programmbibliotheken, die XPath-Anfragen unterstützen, können die Wiederverwendung von Code durch besseres Auffinden eines Codes bzw. von Code-Fragmenten oder Templates verbessert werden.

Anhang:

Listing 1: TestRegistry.java

```

5  import electric.registry.Registry;
   public class TestRegistry
   {
       ... //weiterer Quellcode in dem etwas geändert werden muss
   }

```

Listing 2: TestRegistry.xjava

```

15  <?xml version="1.0" encoding="UTF-8"?>
    <java>
        <import>
            <dot><dot><name>electric</name><name>registry</name></dot><name>Registry</name></dot>
        </import>
        <class>
            <modifiers><public/></modifiers>
            <name>TestRegistry</name>
            <block>
                ...
            </block>
        </class>
25  </java>

```

Listing 3: VariationPointT1.xsl

```

30  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!-- *****
        *** copy template **
        *****-->
    <xsl:template match="*">
        <xsl:copy><xsl:apply-templates/></xsl:copy>
35  </xsl:template>
    <!-- *****
        *** Variation Point Transformation 1 **
        *****-->
    <xsl:template match="import">
        <xsl:if test="dot/name='Registry'">
            <import>
                <dot>
                    <dot><dot><name>org</name><name>apache</name></dot><name>axis</name></dot>
                    <name>client</name></dot><name>Call</name>
                </dot>
            </import>
            <import>
                <dot>
                    <dot><dot><name>org</name><name>apache</name></dot><name>axis</name></dot>
                    <name>client</name></dot><name>Service</name>
                </dot>
            </import>
        </xsl:if>
        <xsl:if test="dot/name!='Registry'">
            <xsl:copy-of select="."/>
        </xsl:if>
        ...
    </xsl:template>
60  </xsl:stylesheet>

```

Listing 4: TestRegistry.xjava (*)

```

<?xml version="1.0" encoding="UTF-8"?>

```

```

<java>
  <import>
    <dot>
      <dot><dot><dot><name>org</name><name>apache</name></dot><name>axis</name></dot>
      <name>client</name></dot><name>Call</name>
    </dot>
  </import>
  <import>
    <dot>
      <dot><dot><dot><name>org</name><name>apache</name></dot><name>axis</name></dot>
      <name>client</name></dot><name>Service</name>
    </dot>
  </import>
  <class>
    <modifiers><public/></modifiers>
    <name>TestRegistry</name>
    <block>
      ...
    </block>
  </class>
</java>

```

Listing 5: TestRegistry.java (*)

```

25 import org.apache.axis.client.Call; //Axis
import org.apache.axis.client.Service;
public class TestRegistry
{
30   ... //weiterer Quellcode in dem etwas geändert wurde
}

```


Patentansprüche

1. Verfahren zur kundenseitigen Anpassung von Software,
 - bei dem eine erste Software (SW) bestehend aus Quelltext SC
 - 5 in eine lauffähige zweite Form (SW*) bestehend aus in einer Meta-Auszeichnungssprache formulierten Variationspunkten VP und/oder kompilierten ByteCode bzw. Binärcode B umgewandelt wird,
 - bei dem die Variationspunkte durch eine Transformation (T)
 - 10 nur in Abhängigkeit von Transformationsregeln TR in einen in der Meta-Auszeichnungssprache formulierten zweiten Code (CodeML*) umgewandelt werden und
 - bei dem dieser zweite Code in der veränderten Software (SW*) entweder einen Variationspunkt VP* bildet oder durch
 - 15 einen Konverter (RCONV) in einen Quellcode SC* und mittels eines Compilers (COMP) in einen Binärcode bzw. ByteCode B* überführt wird, und sich die erste und die zweite Software in ihrem Programmablauf bzw. Programminhalt unterscheiden.
- 20 2. Verfahren nach Anspruch 1, bei dem die erste Software mindestens einen Variationspunkt VP enthält und die Transformationsregeln TR mindestens einen Modifikationsregel für einen Variationspunkt aufweisen.
3. Verfahren nach Anspruch 1 oder 2, bei dem der Modifikationsregel ein Update auf eine neuere Softwareversion bzw. ein Patching anstößt.
4. Verfahren nach Anspruch 1 oder 2,
- 30 bei dem die Modifikation mindestens eines Variationspunktes VP durch die Transformation T und die Kompilierung von VP* speziell zur Laufzeit anstatt davor erfolgt.
5. Verfahren nach einem der vorhergehenden Ansprüche,
- 35 bei dem die Programmiersprache des Quellcodes Java und die Meta-Auszeichnungssprache der Variationspunkte XML ist und

bei dem die Transformation und die Regelbeschreibung mittels XSLT und XSL erfolgt.

6. Anordnung zur kundenseitigen Anpassung von Software,

- 5 - bei der ein erster Konverter (CONV) und Kompiler (COMP) derart vorhanden ist, dass eine erste Software (SW) in einer Meta-Auszeichnungssprache formulierte Variationspunkte VP und kompilierte Bestandteile B umgewandelt wird,
- 10 - bei der ein Prozessor derart vorhanden ist, dass diese Variationtspunkte VP durch eine Transformation T nur in Abhängigkeit von Transformationsregeln TR in einen in der Meta-Auszeichnungssprache formulierten zweiten Code (CodeML*) bzw. VP* umgewandelt werden und
- 15 - bei der ein zweiter Konverter (RCONV) bzw. Kompiler (COMP) derart vorhanden ist, das dieser zweite Code in einer zweiten Software (SW*) zusätzlich als SC* bzw. B* enthalten sein kann, wobei sich die erste und die zweite Software in ihrem Programminhalt und Programmablauf unterscheiden.

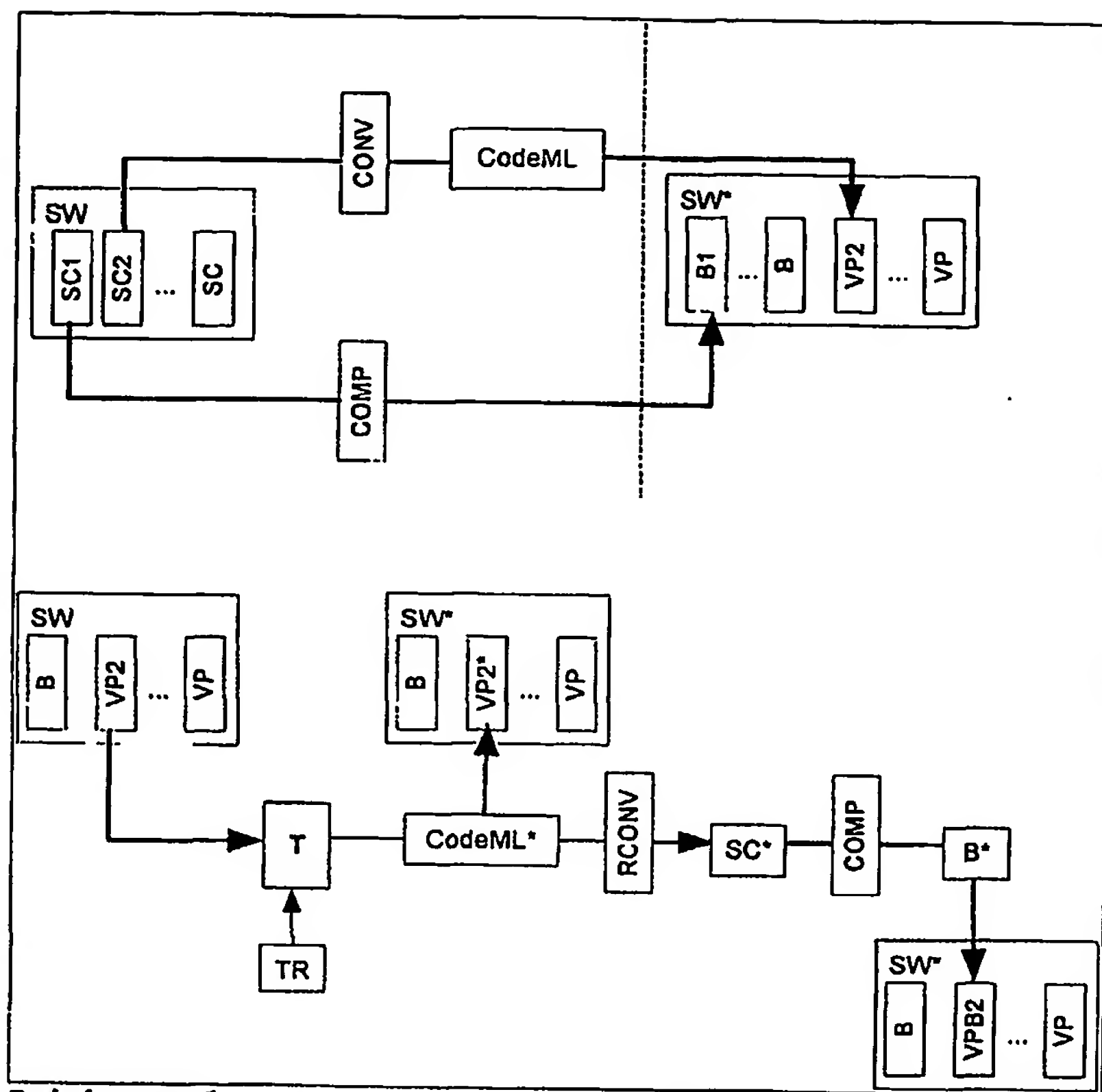
Zusammenfassung

Verfahren und Anordnung zur kundenseitigen Anpassung von
5 Software

Die Erfindung besteht im Wesentlichen darin, dass eine erste
Software mit Variationspunkten VP ausgestattet werden kann,
welche in eine Meta-Auszeichnungssprache formuliert sind, und
10 kundenseitig durch eine Transformation nur in Abhängigkeit
von Transformationsregeln TR, die Modifizierungsregeln für
die Variationspunkte enthalten, in eine zweite Software
umgewandelt werden kann, welche die Variationspunkte VP
entweder in modifizierter Form VP* oder in kompilierter Form
15 VPB* enthält. Die zweite umgewandelt Software unterscheidet
sich dabei von der ersten in ihrem Programmablauf bzw.
Programminhalt.

Zeichnung 1

20



Zeichnung 1